Uke 5

Beholdere 1

21. Februar 2024

agneshb@uio.no

I dag:

- Array og ArrayList
- Generiske klasser
- Lenkelister

ArrayList vs. Array (overblikk)

ArrayList

- Dynamisk størrelse
- Mange innebygde metoder
  - Oversikt over metodene finnes i dokumentasjonen.
- Må inneholde klasser
  - Feks Integer (og ikke int)

Array

- Fast størrelse
- Ikke like mange metoder
- Kan også inneholder primitive typer
  - Feks. Int, men også Integer

ArrayList vs. Array (kode)

Arraylist

```java
//opprette ArrayList:
ArrayList<Integer> tall1 = new ArrayList<>();

//legge til element:
tall1.add(e:1);

//hente element på en indeks:
int i = tall1.get(index:0);

//størrelse:
int str1 = tall1.size();
```

Array

```java
//opprette Array:
int[] tall2 = new int[5];

//legge til element
tall2[0] = 1;

//hente element på indeks:
int j = tall2[0];

//størrelse
int str2 = tall2.length;
//str2 == 5
```

Generiske klasser

- Ønsker ofte at beholdere skal fungere til ulike typer/klasser.
  - Tungvint å lage en type beholder for Integers, og en annen type for String
- Beholderen kan ha en parameter i klassedeklarasjonen.
  - class Beholder<Type> {..}
- Bestemmer typen/klassen beholderen skal ha i det man oppretter en instans av beholderen

Høres det kjent ut?

```
ArrayList<Integer> tall = new ArrayList<>();
ArrayList<String> strings = new ArrayList<>();
```

Generiske klasser

- T er en placeholder for en type.
- Må spesifisere typen T i det vi oppretter en instans av Boks.

- **E** - Element
- **K** - Key
- **N** - Number
- **T** - Type
- **V** - Value

```java
public class Boks<T> {
    private T t;

    public void fyll(T t) {
        this.t = t;
    }

    public T hent() {
        return t;
    }
}
```

```java
Run | Debug
public static void main(String[] args) {

    Boks<String> b = new Boks<String>();
    String tekst = "hallo!";

    b.fyll(tekst);
    System.out.println(b.hent()); //"hallo!"

}
```

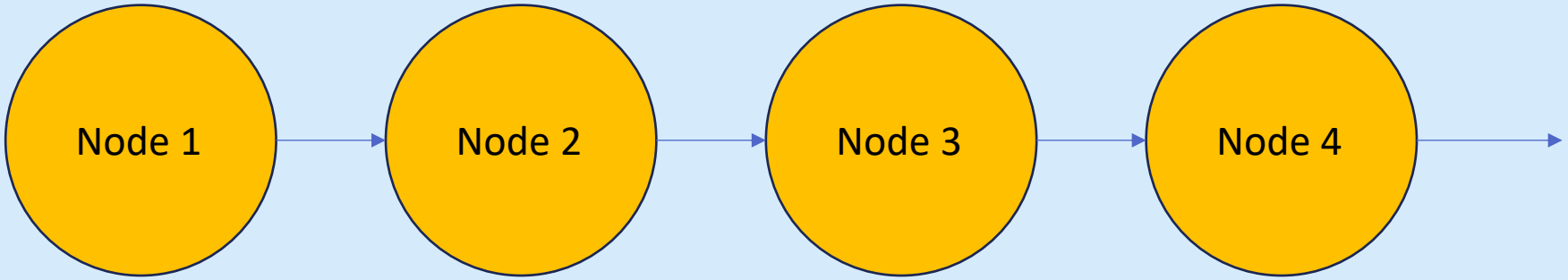
Lenkelister

- Kan implementere egne beholdere.
- En hjelpeklasse Node.
  - Kan hete noe annet, men det er vanlig å kalle den Node.
- Node har en referanse til neste node i lenkelisten.
- Alle noder har dermed en referanse til neste node
  - Unntak: siste node

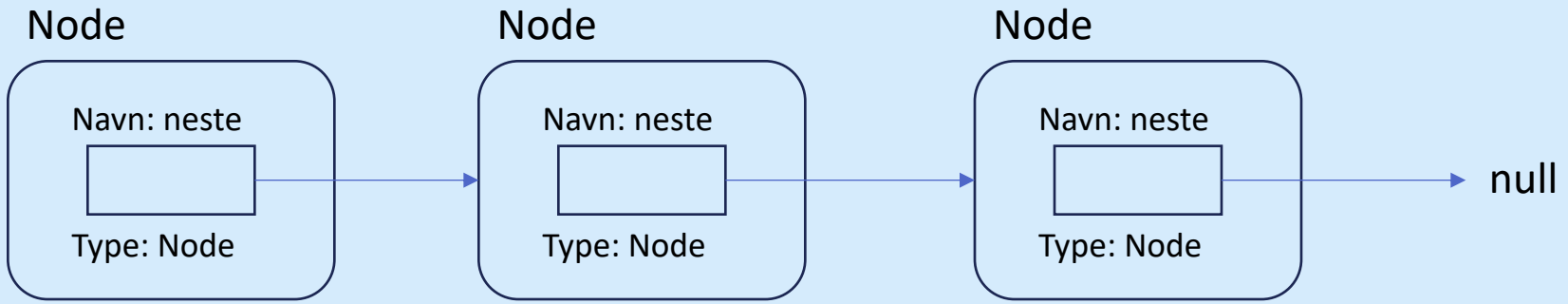
```
class Node {
    Node neste; //= null hvis ingen neste

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```



(kun en visualisering, ikke datastrukturtegning el. l.)




```java
class Node {
    Node neste; //= null hvis ingen neste

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```

Lenkelister

- Må ha en referanse til første noden i lenkelisten for å kunne nå alle elementene i lenkelisten.

```
class Node {
    Node neste; //= null hvis ingen neste

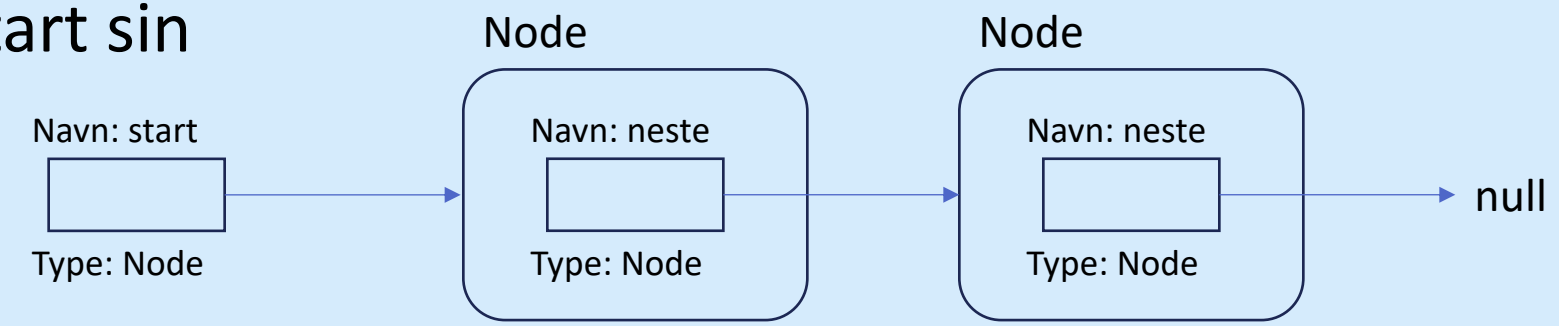
    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```

- Starter med å opprette en startnode (start)
- Oppretter en node som skal komme etter startnoden (nrTo)
- Setter nrTo til å være start sin neste

```
public static void main(String [] args){

    Node start = new Node();
    Node nrTo = new Node();
    start.settNeste(nrTo);
```



Lenkelister

- Node-klassen vi har gått gjennom hittil er ganske ubrukelig da den ikke holder på noe data
  - Kun en peker til neste
- Ofte ønsker vi at en Node inneholder:
  - En peker til neste node (eller null)
  - Data

```
class Node {
    Node neste; //= null hvis ingen neste
    String verdi;

    //Konstruktør:
public Node(String verdi){
        this.verdi = verdi;
    }

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }

    public String hentData(){
        return verdi;
    }
}
```

Lenkelister

```java
class Node {
    Node neste; //= null hvis ingen neste
    String verdi;

    //Konstruktør:
public Node(String verdi){
        this.verdi = verdi;
    }

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }

    public String hentData(){
        return verdi;
    }
}
```

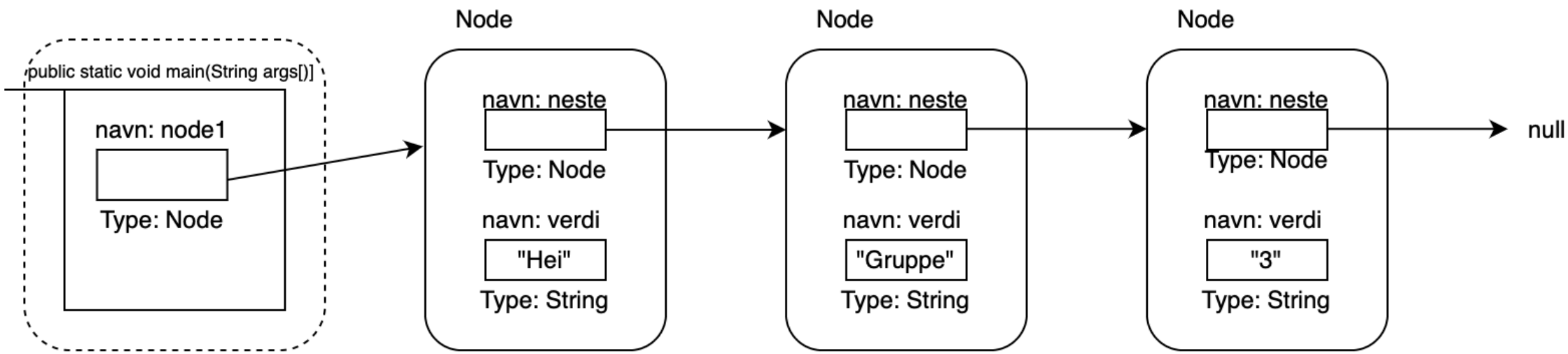
```java
public static void main(String[] args) {

    Node node1 = new Node(verdi:"Hei");
    Node node2 = new Node(verdi:"Gruppe");
    Node node3 = new Node(verdi:"3");

    node1.settNeste(node2);
    node2.settNeste(node3);
```

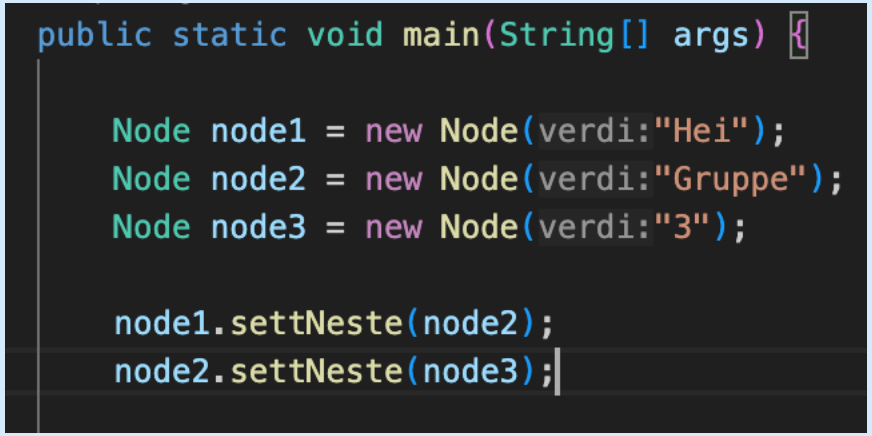
```java
Node node1 = new Node(verdi:"Hei");
node1.settNeste(new Node(verdi:"Gruppe"));
node1.neste.settNeste(new Node(verdi:"3"));
```

Tegne på tavlen


public static void main(String args[])
navn: node1
Type: Node
Node
navn: neste
Type: Node
navn: verdi
"Hei"
Type: String
Node
navn: neste
Type: Node
navn: verdi
"Gruppe"
Type: String
Node
navn: neste
Type: Node
navn: verdi
"3"
Type: String
null

Aksessering

- Her kan vi aksessere node 3 på ulike måter
  - Node1.hentNeste().hentNeste()
  - Node2.hentNeste()



```
Node node1 = new Node(verdi:"Hei");
node1.settNeste(new Node(verdi:"Gruppe"));
node1.neste.settNeste(new Node(verdi:"3"));
```

```
System.out.print(node1.hentData() + " ");
System.out.print(node1.hentNeste().hentData() + " ");
System.out.print(node1.hentNeste().hentNeste().hentData());
```

→ «Hei gruppe 3»

Ulike typer lister/lenkelister

- **Kø** – first in first out (FIFO)
  - Som en vanlig kø i butikken.
- **Stabel/stack** – last in first out (LIFO)
  - Som tallerkener som man stabler oppå hverandre.
- **Prioritetskø** – Den med høyest prioritet først
  - Feks den sykeste pasienten på legevakten.

Jobbe med oppgaver<3

(har lagt ut nettskjema på gruppe 3 -> uke 5)

Seminaroppgaver gruppe 3 -> uke 5

Trix (uke 6)